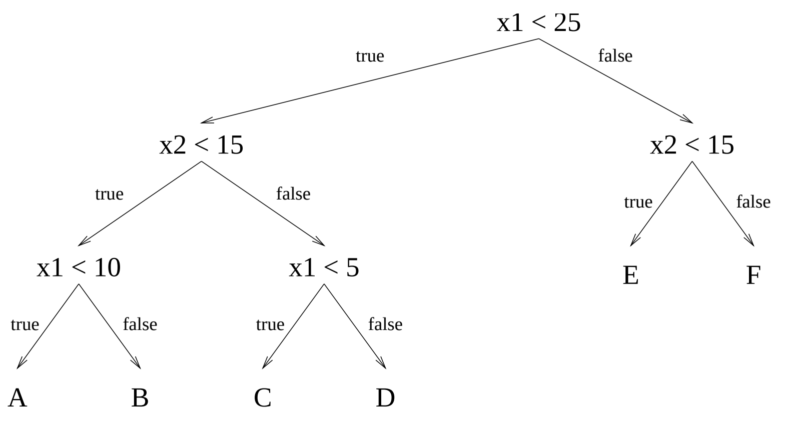
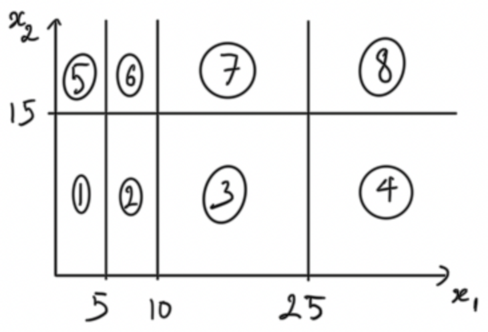
AI534 — Written Assignment 4 —

1. (6 pts) Consider the following decision tree:



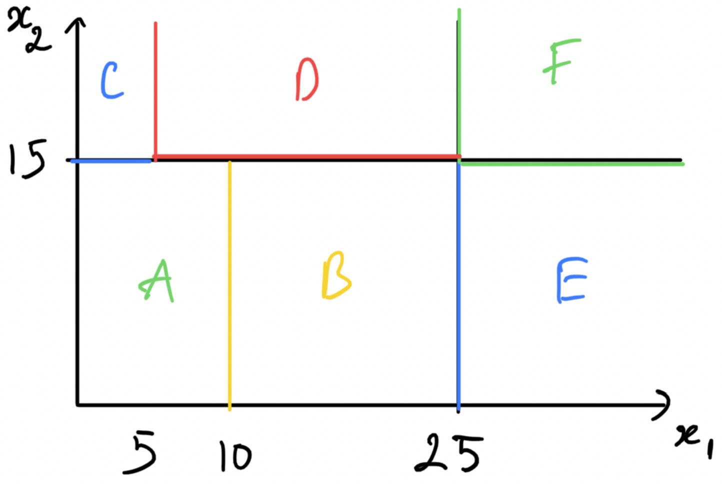
   (a) (2 pts) Draw the decision boundaries defined by this tree. Each leaf of the tree is labeled with a letter. Write this letter in the corresponding region of input space.

   First, let us name the regions divided the boundaries as follows

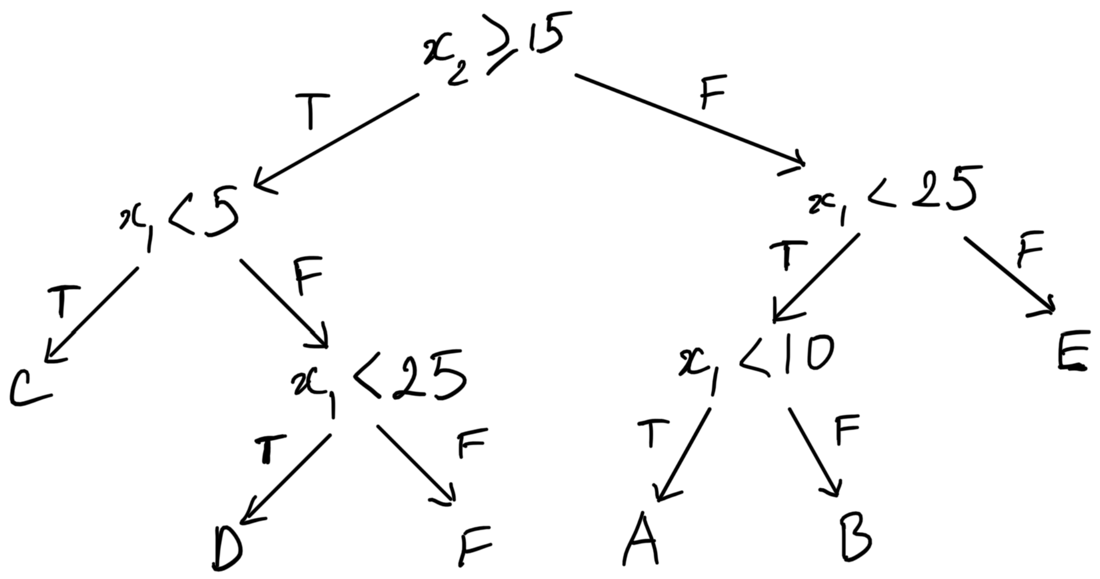


   The areas covered by the letters are:
   A: 1 + 2
   B: 3
   C: 5
   D: 6 + 7
   E: 4
   F: 8

   The decision boundaries are depicted as the colored lines in the following figure



(b) (2 pts) Give another decision tree that is syntactically different but defines the same decision boundaries. This demonstrates that the space of decision trees is syntactically redundant.



(c) (2 pts) How does this redundancy influence learning (does it make it easier or harder to find an accurate tree)?

First, redundancy means that there are some trees having the same decision boundaries, and thus same performance (gini index or information gain) at the leaf nodes.

Second, assuming that the greedy strategy is used for learning because finding the optimal partitioning of the data is an NP-complete problem. The greedy algorithm picks the best combination of feature and threshold (or category in the case of categorical features) for only one node at a time. The discussion boils down to how the algorithm breaks ties. If it breaks ties randomly then its complexity will remain the same regardless of the presence of the redundancy. If it looks ahead for smaller trees, then the complexity will increase by O(TD), where T is the number of ties and D is the maximum depth of the look ahead. The redundancy might create more ties at some nodes (not necessary), thus makes learning harder.

2. (6 pts) In the basic decision tree algorithm (assuming we always create binary splits), we choose the feature/value pair with the maximum information gain as the test to use at each internal node of the

decision tree. Suppose we modified the algorithm to choose at random from among those feature/value combinations that had non-zero mutual information, and we kept all other parts of the algorithm unchanged.

(a) (2 pts) What is the maximum number of leaf nodes that such a decision tree could contain if it were trained on m (distinct) training examples?

In the worst case, every split will split one example from the rest. Hence, the algorithm will result in the binary tree with the height of $m-1$ and m leaf nodes.

(b) (2 pts) What is the maximum number of leaf nodes that a decision tree could contain if it were trained on m training examples using the original maximum mutual information version of the algorithm? Is it bigger, smaller, or the same as your answer to (b)?

On average, the algorithm splits the training examples into two halves. So the average depth would be $O(log_2(m))$, which is shorter than those produced by random split.

(c) (2 pts)How do you think this change (using random splits vs. maximum information mutual information splits) would affect the accuracy of the decision trees produced on average? Why?

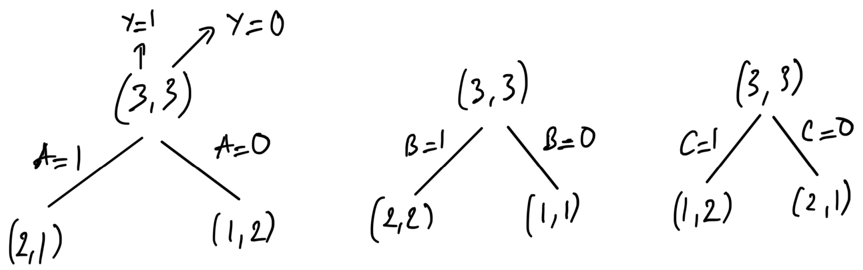
On average, maximum mutual information splits would yield decision trees with better validation accuracy but lower training accuracy. Because random split tend to split training data into small chunks of data, which makes the resulted models prone to overfitting.

3. (8 pts) Consider the following training set:

A	B	C	Y
0	1	1	0
1	1	1	0
0	0	0	0
1	1	0	1
0	1	0	1
1	0	1	1

Learn a decision tree from the training set shown above using the information gain criterion. Show your steps, including the calculation of information gain (you can skip $H(y)$ and just compute $H(y|\mathbf{x})$) of different candidate tests. You can randomly break ties (or better, choose the one that give you smaller tree if you do a bit look ahead for this problem).

First split:



$H(Y) = 1$

$P(A=1) = 0.5, H(Y|A=1) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.918$

$P(A=0) = 0.5, H(Y|A=0) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.918$

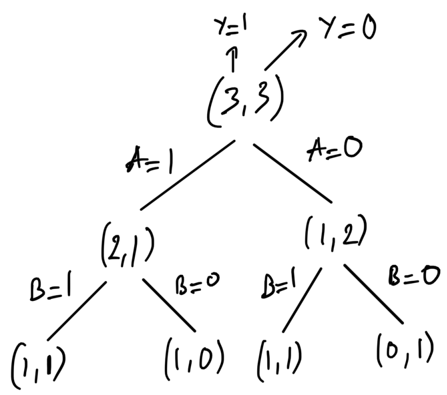
$H(Y|A) = 0.5 \times 0.918 + 0.5 \times 0.918 = 0.918$

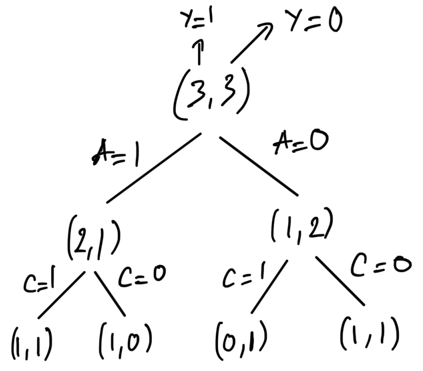
$H(Y|C) = H(Y|A) = 0.5$ *because they split the data in the same way.*

$H(Y|B) = 1$ *because both child nodes have entropy of 1.*

Hence, A is chosen (random tie break with C) for the first split.

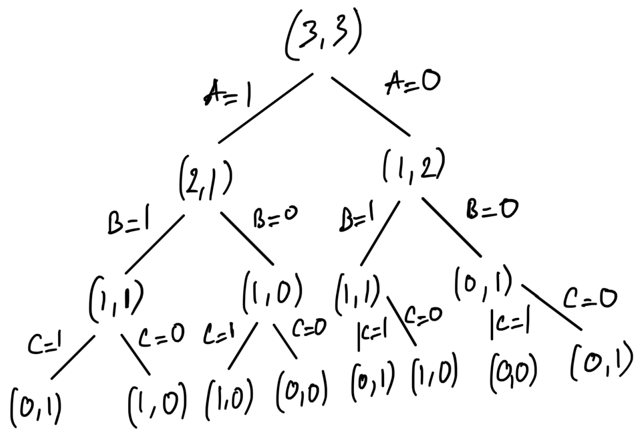
Second split:





$P(A=1,B=1)=\frac{1}{3}, H(Y|A=1,B=1)=1$
$P(A=1,B=0)=\frac{1}{6}, H(Y|A=1,B=0)=0$
$P(A=0,B=1)=\frac{1}{3}, H(Y|A=0,B=1)=1$
$P(A=0,B=0)=\frac{1}{6}, H(Y|A=0,B=0)=0$
$H(Y|A,B)=\frac{1}{3}\times 1+\frac{1}{6}\times 0+\frac{1}{3}\times 1+\frac{1}{6}\times 0=\frac{2}{3}$
$H(Y|A,C)=H(Y|A,B)$ because they split the data in the same way. Hence, B is randomly chosen for the second split.

The third split therefore uses C.



Observe that all of the leaf nodes are either pure or empty. This means that $H(Y|A,B,C)=0$.

4. (5 pts) Please show that in iteration l of Adaboost, the weighted error of h_l on the updated weights D_{l+1} is exactly 50%. In other words, $\sum_{i=1}^{N} D_{l+1}(i)I(h_l(X_i)\neq y_i)=50\%$, where $I(\cdot)$ is the indicator function that takes value 1 if the argument is true. (Hint: start with the condition that, post update, the total weight of correct examples equals the total weight of in-correct examples, i.e., 50% each.)

   First, after the update, the total weight of incorrect examples equals 50% of the total weights.

$$\sum_{i=1}^{N} D_{l+1}(i)I(h_l(X_i)\neq y_i)=\frac{1}{2}\sum_{i=1}^{N} D_{l+1}(i) \qquad (4.1)$$

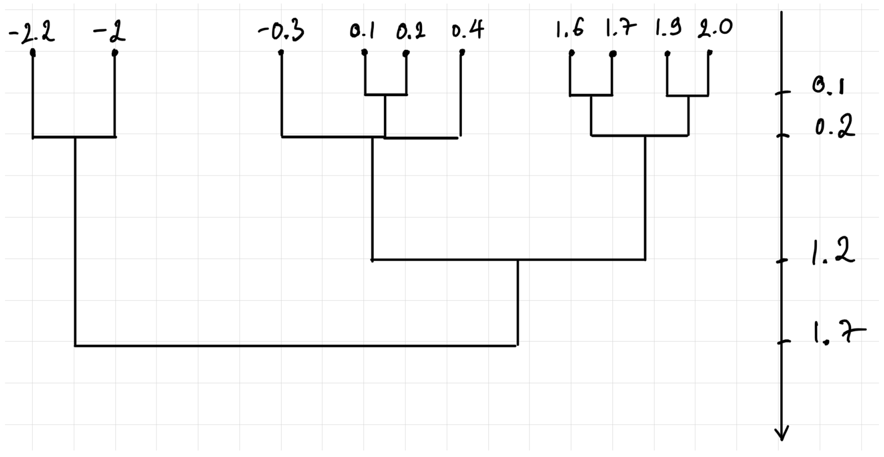
   Second, the weighted error of h_l on the updated weights D_{l+1} can be calculated as follows

$$error(h_l,S,D_{l+1})=\frac{\sum_{i=1}^{N} D_{l+1}(i)I(h_l(X_i)\neq y_i)}{\sum_{i}^{N} D_{l+1}(i)}\overset{(4.1)}{=}\frac{1}{2}$$

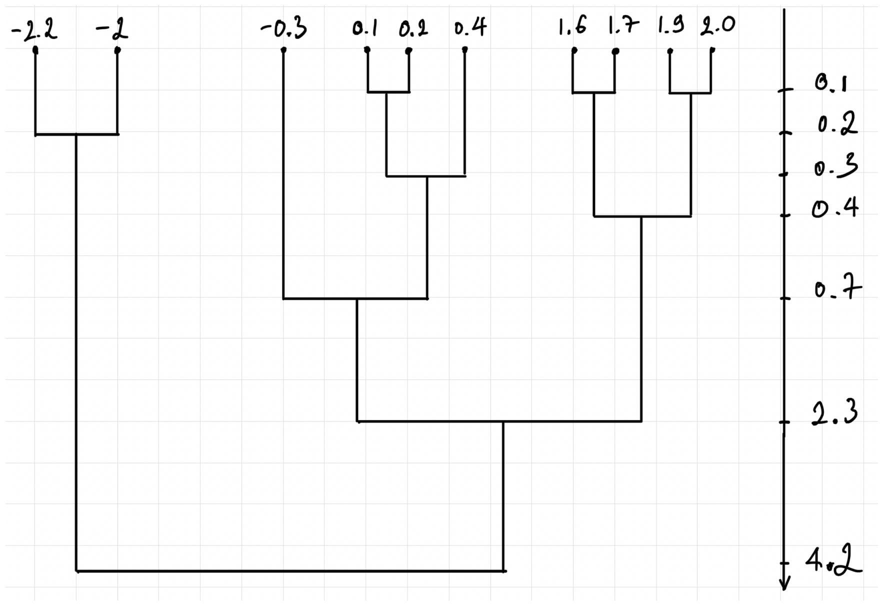
5. **HAC**. (4pts) Create by hand the clustering dendrogram for the following samples of ten points in one dimension.

$$Sample = (-2.2, -2.0, -0.3, 0.1, 0.2, 0.4, 1.6, 1.7, 1.9, 2.0)$$

a. (2pts) Using single link.



b. (2pts) Using complete link



6. (6 pts)Deriving Kmeans for L_1 norm. Consider replacing the distance function used for Kmeans with L_1 norm with the following objective:

$$\min_{\mu_1,\ldots,\mu_K,C_1,\ldots,C_K} \sum_{i=1}^{K} \sum_{x \in C_i} |x - \mu_i|$$

- (3 pts) Show that given fixed cluster assignment $C_1, ..., C_K$, the prototype μ_i that optimizes the above objective can be obtained by taking element-wise median of all the points in cluster i.
  For simplicity, suppose that we have a cluster of data in 1-dimensional space. First, the algorithm randomly picks the initial center μ. The chosen center splits the data into parts, on its left (L) and on its right (R). Let us call the sum of distances between μ and all of the L data points d_L,

that for R data points d_R, and the total distance $d = d_L + d_R$. When we move the center to the left an amount x, $d'_L = d_L - n_L x$, $d'_R = d_R + n_R x$, and $d' = d'_L + d'_R = d + x(n_R - n_L)$. It can be seen that d is minimal when $n_R = n_L$, which makes the center the median of the data points.

- (3 pts) Modify the kmeans algorithm for this L_1 based objective.

```
 Randomly pick k points as centers
Do {
    Assign data points to the cluster whose median is closest to them.
    Re-estimate the centers of the clusters by calculating their medians.
} While (some data points changed membership)
```